基于Dask并行加速的射电干涉成像网格化方法实现*

李姗姗[1]，骆开达[1]，卫守林[1,2]，戴伟[1,2]，梁波[1,2]

1，昆明理工大学信息工程与自动化学院，云南 昆明 650500

2，云南省计算机应用技术重点实验室，云南 昆明 650500

摘要：快速傅里叶变换比傅里叶变换有更好的算法性能，是射电干涉成像基础算法，但因为天线阵列的不规则采样，需使用网格化算法将可见度数据重采样到规则的网格上，才能应用快速傅里叶变换。基于卷积的网格化计算具有计算密集型和迭代型的特点，特别是在处理海量可见度数据的情况下，高性能的网格化计算对加速整个成像过程就显得尤为重要。为了缓解数据处理的压力，在现有处理整块数据和支持多核计算的算法基础上，拓展应用Dask并行计算框架，不仅将数据分块并分配到多线程上，提高数值计算效率，而且动态的分布式任务调度策略优化了网格化的实时处理。实验结果表明多核CPU利用率显著提高，即使增加数据量，也能进一步提升网格化算法性能。分布式任务调度能够将(单)多测量集的网格化弹性缩放到(单)多机系统，充分发挥了集群的规模化优势。

关键词：网格化；干涉成像；分布式计算；并行计算；Dask

中图分类号：P164　　**文献标识码**：A　**文章编号**：xxx

1　引言

射电干涉阵列得到的是非均匀采样的可见度数据，在应用快速傅里叶变换（Fast Fourier Transform, FFT）对可见度数据进行成像前，需使用网格化方法将实际的采样数据重采样到一个均匀划分的网格上。当前网格化主要是使用基于卷积的网格化方法，卷积网格化过程的实质是矩阵相乘，在当数据量较大时，网格化计算是非常耗时。

近年来，天文学家们在提升可见度数据网格化算法性能做了很多研究。其中W-projection算法是目前广泛使用的网格化方法，由于该算法仅校准 *W* 项，并没有校准方向相关效应的 *A* 项，当天线彼此相距较远，*W* 项的尺寸可能会变得很大，使该算法效率低下且占用内存[1]。通过将每个可见度数据的w值投影到邻近的 w 平面的 W-Stacking 算法，可以显著提高网格化性能，但是需要耗费额外的内存[2]。如果考虑方向相关效应，网格化的计算难度将进一步增加，同时修正方向相关效应 *A* 项和 *W* 项被称为AW-projection网格化算法[3]。在数值分析领域，Barnett等人[4]提出基于“半圆指数”卷积核的非均匀傅里叶变换库(Non-uniform Fast Fourier Transform, NUFFT)，将FFT推广到离散化的网格数据中。首次将NUFFT应用到射电天文中的 Nifty-gridder算法，采用共享内存和多线程技术，进一步优化W-Stacking算法。

* 基金项目：科技部SKA专项（2020SKA0110300）；国家自然科学基金（11961141001，U1831204，12063003）；云南省重点研发计划（2018IA054）.

收稿日期：2021-01-20；修订日期：2021-02-01

作者简介：李姗姗，女，硕士研究生，研究方向：计算机技术. Email：2367055935@qq.com

通讯作者：卫守林，男，副教授，研究方向：并行与分布式计算. Email：weishoulin@kust.edu.cn

综上所述，网格化算法的改进和细化都需要计算更多的卷积核，卷积计算占据网格化算法开销的主要部分。虽然采用多核 CPU 和 GPU[5]，可以实现并行计算，提高算法性能，但基于 Python 实现的上述网格化方法主要局限于 NumPy 多维数组计算，难以适应数据的海量性和实时处理需求。近年来数组 Dask.Array 的提出，为超大矩阵的数值计算开辟了新途径。Jamie Farnes 等人[8]采用 Dask 并行框架[9]，配合 Pipeline 技术，测试 LOFAR 数据集，使得原本需要 11 个小时才能完成整个成像流程的串行化代码，缩短至 8 分钟，大大减少了干涉成像所需的时间。本文提出基于 Dask 并行加速的射电干涉可见度数据卷积网格化方法，在并行计算的基础上兼顾系统的弹性缩放，主要特点是以 Dask.Array 矩阵分块存储和计算为核心，封装 Nifty-gridder 卷积网格化算法提供的 Python 接口，采取数据分块和延迟计算，提高了数值计算效率，配合 Dask 的分布式调度策略，实现了网格化算法从单机到集群的迁移。

2 网格化

射电干涉测量方程阐明了可见度数据 V 是天空亮度分布 B 的傅里叶变换，其数学表达式为：

$$K_{pq}=e^{-2\pi i\left(u_{pq}l+v_{pq}m+w_{pq}(n-1)\right)}\#(1)$$

$$V_{pq}=G_p\left(\iint\limits_{lm}\frac{1}{n}K_{pq}A_pBA_q^H\,dldm\right)G_q^H\#(2)$$

其中$(l，m，n)$是观测方向的余弦坐标，$\left(u_{pq}，v_{pq}，w_{pq}\right)$是天线$p$和$q$组成的基线坐标，$G$ 和 A 项分别是琼斯矩阵参数化的方向无关和方向相关效应。在小场近似的条件下，指数中的$w_{pq}(n-1)$趋近于零，可见度和天空亮度近似为二维傅里叶变换关系。由于基线uv轨迹的不规则性，可见度数据并非等间隔离散采样，直接对干涉测量方程进行傅里叶反演的计算代价是非常昂贵的。为了应用 FFT 算法成像，可见度数据必须重新采样到规则化的笛卡尔网格中(Gridding)。

在成像流程图 1 中，不同的光谱频率（即图像通道）测量所得的可见度数据可以独立处理。一个图像通道通常对应于一个或多个数据通道。成像通常从空白的天空模型开始迭代，经过网格化和傅里叶逆变换运算，一个或多个明亮的源可能会掩盖周围微弱的光源，使用 CLEAN 算法提取明亮点源到天空模型中。与网格化相反的过程是对天空模型进行快速傅里叶变换，即从天空模型计算模型的可见度，这被称为去网格化(Degridding)。测量可见度减去模型可见度数据是为了进一步提取微弱光源。重复网格化和去网格化，直到天空模型收敛。

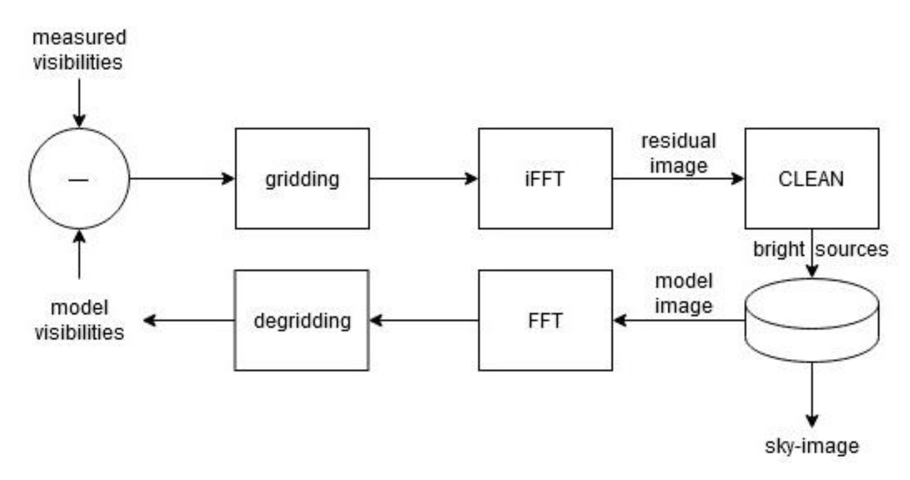


图 1　射电干涉成像流程

Fig. 1　The imaging pipeline of radio interferometry

干涉成像是射电天文中的关键步骤。简单地将可见度数据插值到邻近的网格中会导致严重的伪影，特别是图像混叠(视场外的强光源甚至噪声混叠到视场中)。为了抑制图像伪影的副作用，通常采用可见度数据与网格化函数(卷积核)进行卷积，然后再重采样到网格中，可以提供抗重叠效果。由于卷积核的窗函数特性，使得边界处的图像裁剪误差比中心位置要高出几个数量级，产生了较大的脏图。这需要脏图与修正函数相乘来抵消卷积核产生的误差，从而获得正确的光通量，且该修正函数通常是卷积核的傅里叶逆变换的倒数。相比于 W-Stacking 算法，Nifty-gridder 为提高卷积核的计算精度做了以下改进：(1)沿着 w 轴，对所有的可见度数据网格化到三维uvw空间内，而不是将每个可见度数据的 w 值投影到邻近的 w 平面；(2)改进后的修正函数使脏图的 FFT 和 DFT 之间的差异值最小化（DFT 是无损变换），因此获得更高精度的脏图[10]。

3　卷积网格化实现

3.1　测量集的并行读取和分块

相比于 NumPy.ndarray 数组，Dask.Array 具有如下优势：(1)支持将超大数组分割成许多个 NumPy.ndarray 小数组；(2)采用阻塞算法能在比内存大的数组上支持多核并行计算。此外我们利用 Xarray 实现矩阵的一致性分块(chunksize)，相关字段的数据可以轻易地转化为 Dask.Array 类型。对于(单)多个测量集文件，统一将路径信息放入列表对象中，使用 Dask.Bag 分布式加载，然后按照测量集中的 FIELD_ID 和 DATA_DESC_ID 字段分组，实现并行加载。在本实验中整个数据集划为四个子数据集：(0_0，0_1，0_2 和 0_3)。以子集 0_1 为例，部分重要字段及数据类型如下表 1 所示。

表 1　Xarray 数据集定义的部分相关实验数据

Tab. 1　Xarray dataset definitions for some related experiment data

Dimensions:	(ant: 27, chan: 64, corr: 2, row: 413696, uvw: 3, xyz: 3)

Coordinates:	
ROWID	(row) int32 dask.array<chunksize=(20000,), meta=np.ndarray>
Dimensions without coordinates: ant, chan, corr, row, uvw, xyz	
Data variables:	
ANTENNA1	(row) int32 dask.array<chunksize=(20000,), meta=np.ndarray>
ANTENNA2	(row) int32 dask.array<chunksize=(20000,), meta=np.ndarray>
FLAG	(row, chan, corr) bool dask.array<chunksize=(20000, 64,2),meta=np.ndarray>
DATA	(row, chan, corr) complex64 dask.array<chunksize=(20000, 64, 2), meta=np.ndarray>
UVW	(row, uvw) float64 dask.array<chunksize=(20000, 3), meta=np.ndarray>
WEIGHT	(row, corr) float32 dask.array<chunksize=(20000, 2), meta=np.ndarray>
CHAN_FREQ	(chan) float64 dask.array<chunksize=(64,), meta=np.ndarray>
Attributes:	
FIELD_ID: 0	
DATA_DESC_ID: 1	

3.2 网格化方法的并行实现

分布式计算是解决海量数据的有效途径，Dask 并行计算框架提供了灵活多变的分布式调度方式。由于 Dask 任务调度方式和用户自定义的算法相解耦，用户只需切换调度方式，便可以使算法在(单)多机以多(线)进程的方式弹性扩展，但需要根据算法的特点，选择合理的任务调度方式，以获取最佳的计算性能。本文使用最为复杂的 dask.distributed 调度方式在两台机器节点执行 Nifty-gridder 网格化算法。任务的调度算法(图 2.a)采用多进程的执行方式：多个 MS 文件的物理性分离有利于使用多进程并行读取数据集。MS 文件通常包含多个射电源(即多个 Sub-dataset)，基于子数据集的任务调度更进一步细粒度化整个 Nifty-gridder 网格化的并行流程。使用高阶函数 Dask.Array.blockwise 封装和调用 Nifty-gridder 的 Python 接口，实现了基于子块的并行计算以及协调子块的缩聚和拼接操作(图 2.b)。为避免 Dask.Array 在进程之间的传输成本，数值计算采用多线程的执行方式计算脏图。Nifty-gridder 算法的执行过程如下：

（1）沿着 w 轴确定N_w个采样平面，并均匀分布到 w 轴（从w_0~w_{N_w-1}）；

（2）沿着 w 轴将可见度数据网格化到 w 平面；

（3）初始化$N_x * N_y$的零矩阵I，对每一个$\mathrm{w} = \mathrm{w_i}$平面有：

a) 将每个 w 平面再进行uv网格化，然后执行二维傅里叶逆变换；

b) 裁剪掉 iFFT 图像的外半部，然后乘上$e^{2\pi i w_i(\sqrt{1-l^2-m^2}-1)}$；

c) 将上述结果累加到矩阵I中；

（4）修正函数乘以矩阵I，得到最终的脏图。

(a) Task scheduling on dask.distributed　　(b) Parallel computation in chunk

图 2　分布式任务调度和 Nifty-gridder 网格化算法

Fig. 2　Distributed task scheduling and the Nifty-gridder algorithm

4　实验结果和分析

4.1　实验环境

实验的数据集[1]来源于 2010 年 8 月 23 日，由甚大型 Karl G. Jansky 干涉阵列对超新星遗迹 G055.7+3.4 进行长达 8 小时的观测。该阵列采用 D-型配置，观测频率范围为 1G-2GHz，覆盖所有可用的 L-波段。实验的硬件环境为两台高性能服务器：Intel Xeon CPU E5-2660 v4 CPU @ 3.4GHz 处理器(56 核)，512GB RAM。使用 Common Astronomy Software Applications(CASA 5.6.2)进行数据结果的验证。

4.2 Dask 并行加速和实验结果

以四个子数据集为例，chunksize 设置为 20000 行，经网格化处理生成脏图，使用可见度数据的行数度量数据集的体积，在同一软硬件环境下比较 Dask.Array 和 NumPy 版本 Nifty-gridder 算法的运行时间(单位：秒)，实验结果如下：

表 2　Nifty-gridder 网格化执行时间的比较(Dask.Array vs. NumPy)

[1] http://casa.nrao.edu/Data/EVLA/SNRG55/SNR_G55_10s.calib.tar.gz

Tab.2 The execution times of the Nifty-gridder compared Dask.Array to NumPy

Sub-dataset	Volume(row)	Execution Time (Dask.Array)		Execution Time (NumPy)	Speedup Ratio
		CPU Time(s)	Wall Time(s)		
0_0	39274	2.8	2.68	1.57	0.59
0_1	413696	38.5	4.95	16.18	3.27
0_2	412696	35.9	4.86	14.26	2.93
0_3	414974	39.4	4.95	16.16	3.26

注：Speedup Ratio= Execution Time (NumPy) / Wall Time(s)，且 Wall Time 为程序的实际执行时间。

从表 2 可知，基于 Dask.Array 改进的 Nifty-gridder 算法，其 CPU Time 均大于 Wall Time，说明对于计算密集型问题，使用多核计算并行效果显著，明显降低程序的运行时间。以 0_0 和 0_1 数据集的对比分析为例：即使将可见度数据体积增大 10.5 倍(≈413696/39274)，相应的执行时间 Wall Time 仅仅增加 1.85 倍(≈4.95/2.68)，且加速比进一步提高。然而 Dask.Array 是在 NumPy 的基础上增加了一层复杂的设计，对于较小的数据体积(0_1 数据集约占用 40MB)，NumPy 可能是正确的选择，相反，这恰恰说明了 Dask.Array 适宜处理超大型矩阵的数值计算。

Dask 允许跨集群提交 Python 函数以实现基于任务的并行，从而生成大量可以监视、控制和计算的 Future 对象。对于复杂的程序处理流程，动态的可视化监控有助于了解算法的性能瓶颈，实验执行过程中的实时性能监控如图 3 所示。



图 3 网格化流程中任务流的实时状态

Fig. 3 The real-time state of task stream in the gridding process

为了说明 Dask 并行框架的优越性，通过增加测量集的输入量和限定每台机器内存占有量并确保实验环境一致。从系统资源利用率角度分析并比较基于 Dask.Array 和 NumPy 的 Nifty-gridder 算法性能。由图 4 可知，无论是资源利用率的峰值和平均值，相比于 NumPy

版本，Dask.Array 类型的网格化算法明显占有更低 CPU 利用率和内存占有率，但却能获得更快的网格化执行时间(见表 2)。主要是因为 Dask.Array 数组是采取分块加载和延迟计算，尚不具备计算条件的子块会驻留磁盘，以节约系统资源，而满足计算条件的子块则被送入内存并行执行，相反 NumPy 数组必须全部加载到内存，导致较高内存的持有率。

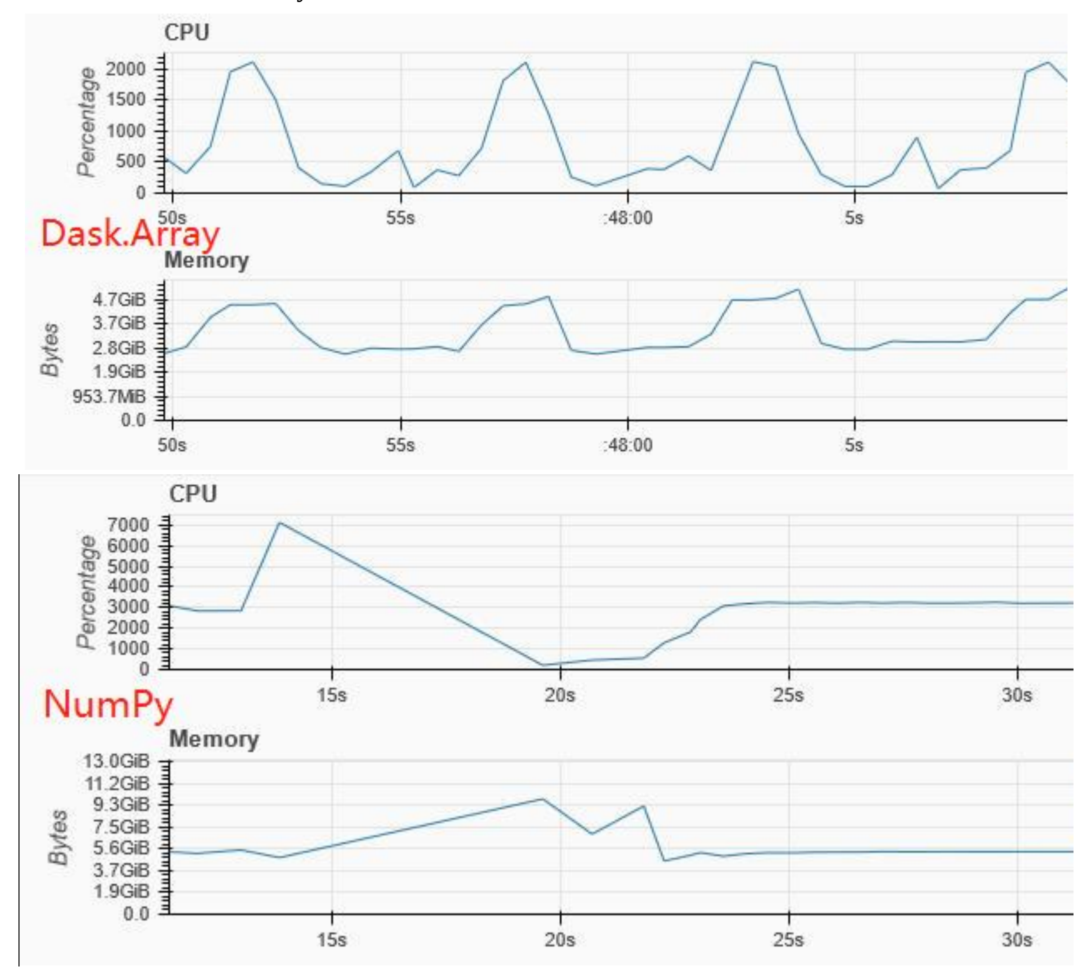


图 4　网格化流程中 CPU 和内存的使用情况对比(Dask.Array vs. NumPy)

Fig. 4　CPU and memory usage in the gridding process compared Dask.Array to NumPy

为了进一步验证代码的正确性，使用标准的 CASA 软件对该数据集进行成像[2]，生成的脏图(图 5 左)与实验结果(图 5 右)进行对比，两幅灰度图中的灰白色点代表观测源，可以发现正确识别出射电源的分布位置。

[2] https://casaguides.nrao.edu/index.php?title=VLA_CASA_Imaging-CASA5.7.0

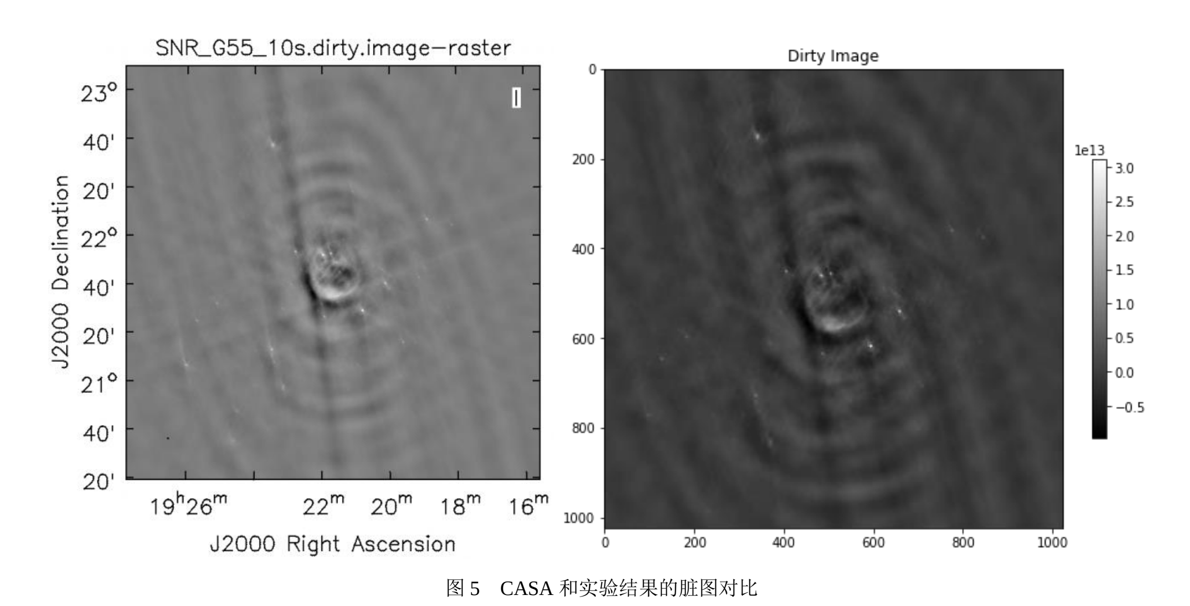


图 5　CASA 和实验结果的脏图对比

Fig. 5　Comparison of dirty image from CASA and experimental result

5　总结

高性能分布式并行计算已成为射电干涉成像过程中应对高分辨率和大视场干涉阵列产生的海量数据的必要方法。可见度数据的网格化和去网格化是成像的重要组成部分，网格化并行加速无疑对于提高整个成像过程速度有重要意义。本文使用了开源的 Dask 分布式计算框架结合 Nifty-gridder 实现了测量集的分布式加载和并行网格化加速过程，充分发挥了集群的规模化优势，提高了多核 CPU 利用率。干涉成像过程中包含多个复杂的处理流程，都涉及矩阵的数值计算，而 Dask.Array 可以高效地处理多维超大矩阵的数值计算，因此下一步的工作考虑基于 Dask 实现去网格化、校准、成像等算法的并行加速。

参考文献

[1] Cornwell T J, Golap K, Bhatnagar S. W projection: A new algorithm for wide field imaging with radio synthesis arrays[C]//Astronomical Data Analysis Software and Systems XIV. 2005, 347: 86.

[2] Cornwell T J, Voronkov M A, Humphreys B. Wide field imaging for the square kilometre array[C]//Image Reconstruction from Incomplete Data VII. International Society for Optics and Photonics, 2012, 8500: 85000L.

[3] Bhatnagar S , Cornwell T J , Golap K , et al. Correcting direction-dependent gains in the deconvolution of radio interferometric images[J]. Astronomy & Astrophysics, 2008, 487(1):419-429.

[4] Barnett A H, Magland J, af Klinteberg L. A parallel nonuniform fast fourier transform library based on an “exponential of semicircle" kernel[J]. SIAM Journal on Scientific Computing, 2019, 41(5): C479-C504.

[5] Veenboer B, Petschow M, Romein J W. Image-domain gridding on graphics processors[C]//2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS). IEEE, 2017: 545-554.

[6] Merry, B. Faster GPU-based convolutional gridding via thread coarsening[J]. Astronomy & Computing, 2016, 16:140-145.

[7] 冯勇,王锋,邓辉,等. 基于 OpenCL 的射电干涉阵成像网格化算法实现[J].天文研究与技术,2019,16(01):8-15. Feng Yong, Wang Feng, Deng Hui, et al. Implementation of Gridding Algorithm for Radio Interferometric Imaging Based on OpenCL[J]. Astronomical Research & Technology, 2019,16(01):8-15.

[8] Farnes J, Mort B, Dulwich F, et al. Science pipelines for the square kilometre array[J]. Galaxies, 2018, 6(4): 120.

[9] Rocklin M. Dask: Parallel computation with blocked algorithms and task scheduling[C]//Proceedings of the 14th python in science conference. Austin, TX: SciPy, 2015, 126.

[10] Ye H, Gull S F, Tan S M, et al. Optimal gridding and degridding in radio interferometry imaging[J]. Monthly Notices of the Royal Astronomical Society, 2020, 491(1): 1146-1159.

A distributed gridding implementation method for radio interferometric visibilities based on DASK

Li Shanshan[1], Luo Kaida[1], Wei Shoulin[1,2], Dai Wei[1,2], Liang Bo[1,2]

(1, Faculty of Information Engineering and Automation, Kunming University of Science and Technology, Kunming 650500, China;

2, Key Laboratory of Applications of Computer Technology of the Yunnan Province, Kunming 650500, China)

Abstract: Fast Fourier Transform (FFT) has better performance than Discrete Fourier Transform, which is the fundamental imaging algorithm of radio interferometry. However, because of the irregular sampling of antenna array, it is necessary to use gridding algorithms to resample visibilities to regular grids, so that FFT can be applied. The convolutional gridding in radio interferometric imaging is characterized by intensive and iterative computations. Especially in the case of massive visibility data processing, high-performance gridding computing is particularly important to accelerate the whole imaging process. In order to alleviate the pressure of data processing, the DASK parallel computing framework is extended and applied on the existing gridding algorithm which supports multi-core parallelism but processes whole blocks of data. Not only can the data be partitioned and distributed to multiple threads to improve the efficiency of numerical computation, but also the dynamic distributed task scheduling strategy can optimize the real-time workflow of gridding. The experimental results show that the multi-core utilization rate is significantly improved and the performance of gridding algorithm can be further enhanced even if the volume of visibility is increased. Distributed task scheduling can flexibly scale the gridding task of (single) multi-measurement set to (single) multi-machine system, which gives full play to the scale advantage of clustering.

Key words: Gridding; Interferometric imaging; Distributed Computing; Parallel Computing; Dask